Shōyu

SMART CONTRACT AUDIT

ZOKYO.

Aug 30th, 2021 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges

SCORE

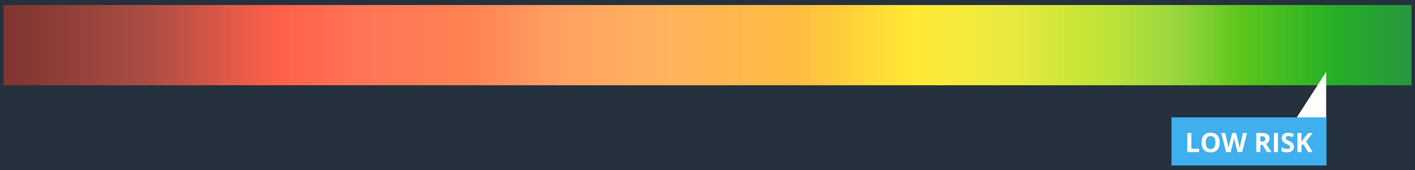
95

TECHNICAL SUMMARY

This document outlines the overall security of the Shoyu smart contracts, evaluated by Zokyo's Blockchain Security team.

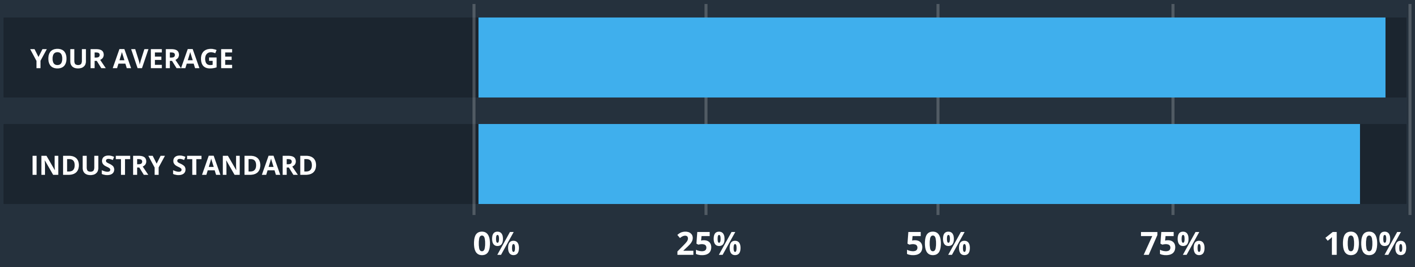
The scope of this audit was to analyze and document the Shoyu smart contract codebase for quality, security, and correctness.

Contract Status



There were 2 critical issues found during the audit.

Testable Code



The testable code is 97.75%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that’s able to withstand the Ethereum network’s fast-paced and rapidly changing environment, we at Zokyo recommend that the Shoyu team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Shoyu repository.

Repository:
https://github.com/sushiswap/shoyu/commit/d5b1aac3b58c900de8c04169f7b754abc66b598a

Last commit:
cd6b18e839b1f1010b8ef71c8131deb11e52c5d1

Contracts:

- NFT721.sol
- NFT1155.sol
- ERC721Exchange.sol
- ERC1155Exchange.sol
- TokenFactory.sol

Depencies:

- BaseExchange.sol
- BaseNFT721.sol
- INFT721.sol
- INFT1155.sol
- BaseNFT1155.sol
- ITokenFactory.sol
- ProxyFactory.sol

Requirements:
https://drive.google.com/drive/folders/1-TEPnGe2I7PToJ47C5WUlT_RuiGdrRfl?usp=sharing

Throughout the review process, care was taken to ensure that the token contract:

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1. Due diligence in assessing the overall code quality of the codebase.
2. Cross-comparison with other, similar smart contracts by industry leaders.
3. Testing contract logic against common and uncommon attack vectors.
4. Thorough, manual review of the codebase, line-by-line.

SUMMARY

During the audit, Zokyo team has found a couple of issues. Among them there were two critical issues, which were successfully resolved by the Shoyu development team.
Hence, these two issues are not bearing a risk as of now.

The contracts are in good condition and bear no security or operational risk. They are fully production ready.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

● Critical

The issue affects the ability of the contract to compile or operate in a significant way.

● High

The issue affects the ability of the contract to compile or operate in a significant way.

● Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

● Low

The issue has minimal impact on the contract’s ability to operate.

● Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

Wrong permission in NFT if deployed via factory

CRITICAL | RESOLVED

When the function createNFT721 is called msg.sender is the address of X user, and this address passed as parameter to NFT contract as owner. But msg.sender in NFT721 constructor is the address of the factory, and in the initialize function contract trying to call the function with onlyOwner modifier. Since msg.sender in initialize is factory and owner is msg.sender of factory call (in my case user) it fails. This is also relevant for NFT1155.

Recommendation:
Change onlyOwner modifier to onlyOwnerOrFactory with changing of require in ti. Or provide access for factory to execute setRoyaltyFeeRecipient and setRoyaltyFee functions.

Re-audit:
Fixed, now factory owner passed by parameter of factory call.

Exceeding of gas limit

CRITICAL | RESOLVED

Deploying TokenFactory requires 17,158,715 gas, since ethereum mainnet gas limit is 15 million it will be reverted. Theoretically it can be executed because when contracts are deploying blockchain provides a separate block for each contract with their own limits, but I can't prove it with tests (it doesn't work), so I can't guarantee that it will be deployed. Also on binance smart chain limit is around 80 millions.

Recommendation:
Change TokenFactory constructor to be in the 15 million gas range. Or provide information about on which network contracts will be deployed.

Re-audit:
Fixed, now contract deployment is under 15,000,000 gas.

Use OpenZeppelin's safe math library

LOW | UNRESOLVED

Since the there are chances of underflow/overflow occurrences during mathematical operations, it is recommended to use openzeppelin safemath.sol library for all mathematical operations: openzeppelin-contracts/SafeMath.sol at master · OpenZeppelin/openzeppelin-contracts (github.com)

Comment from Shoyu team:
For openzeppelin SafeMath issue, it doesn't apply to our case since we use solc 0.8.3.

Correcting the require statements to be more precise

LOW | NOT VALID

Since the transactions would fail if the require condition is not satisfied, So it is recommended to add a statement which justifies the reason for failing of the transaction.

Recommendation:

1) In ProxyFactory.sol file, in _createProxy() function, replace

```
require(success, "SHOYU: CALL_FAILURE")
```

with

```
require(success, "SHOYU: FAILED TO CALL THE REQUESTED FUNCTION")
```

2) In the Tokenfactory.sol file, in setBaseURI721(), length of the argument string could be checked to make sure that uri cannot be zero before updating the state.

3) In the Tokenfactory.sol file, in setBaseURI1155(), length of the argument string could be checked to make sure that uri cannot be zero before updating the state.

4) In Tokenfactory.sol file, in setProtocolFeeRecipient(), replace

```
require(protocolFeeRecipient != address(0), "SHOYU: INVALID_FEE_RECIPIENT")
```

with

```
require(protocolFeeRecipient != address(0), "SHOYU: INVALID_PROTOCOL_FEE_RECIPIENT")
```

5) In Tokenfactory.sol file, in setOperationalFeeRecipient(), replace

```
require(operationalFeeRecipient != address(0), "SHOYU: INVALID_RECIPIENT")
```

with

```
require(operationalFeeRecipient != address(0), "SHOYU: INVALID_OPERATIONAL_RECIPIENT")
```

6) In Tokenfactory.sol file, in setOperationalFee(), replace

```
require(operationalFee <= MAX_OPERATIONAL_FEE, "SHOYU: INVALID_FEE")
```

with

```
require(operationalFee <= MAX_OPERATIONAL_FEE, "SHOYU: OPERATIONAL FEE IS GREATER
THAN THE MAXIMUM ALLOWED OPERATIONAL FEE")
```

7) In the Tokenfactory.sol file, in isNFT721(), check could be made such that the passed query is not zero before calling the _isProxy() function.

8) In the Tokenfactory.sol file, in isNFT1155(), check could be made such that the passed query is not zero before calling the _isProxy() function.

9) In the Tokenfactory.sol file, in createSocialToken(), check whether the arguments name and symbol strings are Non-empty before executing the remaining code.

10) In the Tokenfactory.sol file, in isSocialToken(), check could be made such that the argument query is not zero before calling the _isProxy() function.

11) In Tokenfactory.sol file, in setTags721(), replace

```
require(IBaseNFT721(nft).ownerOf(tokenId) == msg.sender, "SHOYU: FORBIDDEN")
```

with

```
require(IBaseNFT721(nft).ownerOf(tokenId) == msg.sender, "SHOYU: ONLY OWNER OF THE
TOKEN CAN SET THE TAGS")
```

12) In Tokenfactory.sol file, in setTags1155(), replace

```
require(IBaseNFT1155(nft).balanceOf(msg.sender, tokenId) > 0, "SHOYU: FORBIDDEN")
```

with

```
require(IBaseNFT1155(nft).balanceOf(msg.sender, tokenId) > 0, "SHOYU: BALANCE SHOULD BE
MORE THAN ZERO")
```

13) In ERC721Initializable.sol, in _transfer(), replace

```
require(ERC721Initializable.ownerOf(tokenId) == from, "ERC721: transfer of token that is not own")
```

with

```
require(ERC721Initializable.ownerOf(tokenId) == from, "ERC721: Cannot transfer the token that is not own")
```

14) In ERC721Initializable.sol, in _transfer(), replace

```
require(to != address(0), "ERC721: transfer to the zero address")
```

with

```
require(to != address(0), "ERC721: to address cannot be zero")
```

15) In ERC721Initializable.sol, in safeTransferFrom(), replace

```
require(_isApprovedOrOwner(msg.sender, tokenId), "SHOYU: FORBIDDEN")
```

with

```
require(_isApprovedOrOwner(msg.sender, tokenId), "SHOYU: CALLER IS NEITHER OWNER NOR APPROVED")
```

16) In BaseNFT721.sol file, in parkTokenIds(), replace

```
require(owner() == msg.sender, "SHOYU: FORBIDDEN")
```

with

```
require(owner() == msg.sender, "SHOYU: CALLER IS NOT THE OWNER")
```

17) In BaseNFT721.sol file, in mint(), replace

```
require(_factory == msg.sender || owner() == msg.sender, "SHOYU: FORBIDDEN")
```

with

```
require(_factory == msg.sender || owner() == msg.sender, "SHOYU: CALLER IS NEITHER THE
OWNER NOR THE TOKEN CREATOR")
```

18) In BaseNFT721.sol file, in mintBatch(), replace

```
require(owner() == msg.sender, "SHOYU: FORBIDDEN")
```

with

```
require(owner() == msg.sender, "SHOYU: CALLER IS NOT THE OWNER")
```

19) In BaseNFT721.sol file, in burn() & burnBatch(), replace

```
require(ownerOf(tokenId) == msg.sender, "SHOYU: FORBIDDEN")
```

with

```
require(ownerOf(tokenId) == msg.sender, "SHOYU: CALLER IS NOT THE OWNER OF THE TOKEN")
```

20) In BaseNFT721.sol file, in permit(), replace

```
require(spender != owner, "SHOYU: NOT_NECESSARY")
```

with

```
require(spender != owner, "SHOYU: OWNER OF THE TOKEN CANNOT BE THE SPENDER")
```

21) In NFT721.sol file, in setRoyaltyFeeRecipient(), replace

```
require(royaltyFeeRecipient != address(0), "SHOYU: INVALID_FEE_RECIPIENT")
```

with

```
require(royaltyFeeRecipient != address(0), "SHOYU: INVALID_ROYALTY_FEE_RECIPIENT")
```

22) In NFT721.sol file, in setRoyaltyFee(), replace

```
require(royaltyFee <= ITokenFactory(_factory).MAX_ROYALTY_FEE(), "SHOYU: INVALID_FEE")
```

with

```
require(royaltyFee <= ITokenFactory(_factory).MAX_ROYALTY_FEE(), "SHOYU: ROYALTY FEE
CANNOT BE MORE THAN THE SET MAX_ROYALTY_FEE VALUE")
```

23) In ERC1155Initializable.sol, in safeTransferFrom(), replace

```
require(from == msg.sender || isApprovedForAll(from, msg.sender), "SHOYU: FORBIDDEN")
```

with

```
require(from == msg.sender || isApprovedForAll(from, msg.sender), "SHOYU: CALLER IS NEITHER
OWNER NOR APPROVED")
```

24) In ERC1155Initializable.sol, in safeBatchTransferFrom(), replace

```
require(from == msg.sender || isApprovedForAll(from, msg.sender), "SHOYU: FORBIDDEN")
```

with

```
require(from == msg.sender || isApprovedForAll(from, msg.sender), "SHOYU: CALLER IS NEITHER
OWNER NOR APPROVED")
```

25) In BaseNFT1155.sol file, in mint(), replace

```
require(_factory == msg.sender || owner() == msg.sender, "SHOYU: FORBIDDEN")
```

with

```
require(_factory == msg.sender || owner() == msg.sender, "SHOYU: CALLER IS NEITHER THE
OWNER NOR THE TOKEN CREATOR")
```

26) In BaseNFT1155.sol file, in mintBatch(), replace

```
require(owner() == msg.sender, "SHOYU: FORBIDDEN")
```

with

```
require(owner() == msg.sender, "SHOYU: CALLER IS NOT THE OWNER")
```

27) In BaseNFT1155.sol file, in permit(), replace

```
require(block.timestamp <= deadline)
```

with

```
require(block.timestamp <= deadline, "SHOYU: DEADLINE EXPIRED")
```

28) In NFT1155.sol file, in setRoyaltyFeeRecipient(), replace

```
require(royaltyFeeRecipient != address(0), "SHOYU: INVALID_FEE_RECIPIENT")
```

with

```
require(royaltyFeeRecipient != address(0), "SHOYU: INVALID_ROYALTY_FEE_RECIPIENT")
```

29) In NFT1155.sol file, in setRoyaltyFee(), replace

```
require(royaltyFee <= ITokenFactory(_factory).MAX_ROYALTY_FEE(), "SHOYU: INVALID_FEE")
```

with

```
require(royaltyFee <= ITokenFactory(_factory).MAX_ROYALTY_FEE(), "SHOYU: ROYALTY FEE
CANNOT BE MORE THAN THE SET MAX_ROYALTY_FEE VALUE")
```

Could make code more efficient

INFORMATIONAL | UNRESOLVED

In ReentrancyGuardInitializable.sol, on lines 35 & 36, the private variables _NOT_ENTERED & _ENTERED could be declared as boolean types instead of unit256:

```
uint256 private constant _NOT_ENTERED = 1;
uint256 private constant _ENTERED = 2;
```

Recommendation:
Replace

```
uint256 private constant _NOT_ENTERED = 1;
uint256 private constant _ENTERED = 2;
```

with

```
bool private constant _NOT_ENTERED =true;
bool private constant _ENTERED =true;
```

And also change the logic according to the changes made.

	NFT721.sol	NFT1155.sol	TokenFactory.sol
Re-entrancy	Not affected	Not affected	Not affected
Access Management Hierarchy	Not affected	Not affected	Not affected
Arithmetic Over/Under Flows	Not affected	Not affected	Not affected
Unexpected Ether	Not affected	Not affected	Not affected
Delegatecall	Not affected	Not affected	Not affected
Default Public Visibility	Not affected	Not affected	Not affected
Hidden Malicious Code	Not affected	Not affected	Not affected
Entropy Illusion (Lack of Randomness)	Not affected	Not affected	Not affected
External Contract Referencing	Not affected	Not affected	Not affected
Short Address/ Parameter Attack	Not affected	Not affected	Not affected
Unchecked CALL Return Values	Not affected	Not affected	Not affected
Race Conditions / Front Running	Not affected	Not affected	Not affected
General Denial Of Service (DOS)	Not affected	Not affected	Not affected
Uninitialized Storage Pointers	Not affected	Not affected	Not affected
Floating Points and Precision	Not affected	Not affected	Not affected
Tx.Origin Authentication	Not affected	Not affected	Not affected
Signatures Replay	Not affected	Not affected	Not affected
Pool Asset Security (backdoors in the underlying ERC-20)	Not affected	Not affected	Not affected

	ERC721Exchange.sol	ERC1155Exchange.sol
Re-entrancy	Not affected	Not affected
Access Management Hierarchy	Not affected	Not affected
Arithmetic Over/Under Flows	Not affected	Not affected
Unexpected Ether	Not affected	Not affected
Delegatecall	Not affected	Not affected
Default Public Visibility	Not affected	Not affected
Hidden Malicious Code	Not affected	Not affected
Entropy Illusion (Lack of Randomness)	Not affected	Not affected
External Contract Referencing	Not affected	Not affected
Short Address/ Parameter Attack	Not affected	Not affected
Unchecked CALL Return Values	Not affected	Not affected
Race Conditions / Front Running	Not affected	Not affected
General Denial Of Service (DOS)	Not affected	Not affected
Uninitialized Storage Pointers	Not affected	Not affected
Floating Points and Precision	Not affected	Not affected
Tx.Origin Authentication	Not affected	Not affected
Signatures Replay	Not affected	Not affected
Pool Asset Security (backdoors in the underlying ERC-20)	Not affected	Not affected

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security team

As part of our work assisting Shoyu in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the Shoyu contract requirements for details about issuance amounts and how the system handles these.

Code Coverage

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
contracts\	97.75	82.43	100.00	97.69	
ERC1155Exchange.sol	92.31	50.00	100.00	91.67	35
ERC721Exchange.sol	92.31	50.00	100.00	91.67	35
NFT1155.sol	100.00	75.00	100.00	100.00	
NFT721.sol	97.30	71.43	100.00	97.14	92
TokenFactory.sol	98.86	90.41	100.00	98.86	86
All files	**97.75**	**82.43**	**100.00**	**97.69**	

Test Results

```
Contract: TokenFactory
  Setup
    ✓ should deploy (1847ms)
  On success
    Getters
    ✓ operational fee info (173ms)
  Setters
    ✓ deployer (397ms)
    ✓ baseURI721 (393ms)
    ✓ baseURI1155 (355ms)
    ✓ protocol recipient (352ms)
    ✓ operational fee/recipient (630ms)
    ✓ strategy whitelisted (420ms)
    ✓ upgrade NFT721 target (513ms)
    ✓ upgrade NFT1155 target (510ms)
    ✓ upgrade NFT721 exchange target (295ms)
    ✓ upgrade NFT1155 exchange target (237ms)
    ✓ upgrade social token (567ms)
    ✓ domain separator (144ms)
  NFT721
    ✓ should deploy via factory [multiple tokens mint] (1359ms)
    ✓ should deploy via factory [single token mint] (1148ms)
    ✓ should check NFT information (503ms)
    ✓ should mint NFT (592ms)
    ✓ should mint NFT with tags (640ms)
    ✓ should set tag (277ms)
    ✓ should call supportsInterface (154ms)
    ✓ should set royalty recipient/fee (688ms)
    ✓ should call internal transfer (442ms)
  NFT1155
    ✓ should deploy via factory (936ms)
    ✓ should check NFT information (437ms)
    ✓ should mint NFT (550ms)
    ✓ should mint NFT with tags (479ms)
    ✓ should set tag (240ms)
    ✓ should call supportsInterface (81ms)
    ✓ should set royalty recipient/fee (629ms)
    ✓ should call internal transfer (359ms)
  NFT721Exchange
    ✓ should call DOMAIN_SEPARATOR (123ms)
    ✓ should check factory (146ms)
    ✓ should can trade (319ms)
    ✓ should call internal transfer (626ms)
```

```
        NFT1155Exchange
          ✓ should call DOMAIN_SEPARATOR (125ms)
          ✓ should check factory (182ms)
          ✓ should can trade (455ms)
          ✓ should call internal transfer (717ms)
        SocialToken
          ✓ should deploy via factory (747ms)
      On reverts
        ✓ should NOT set operational fee bigger then max fee value (962ms)
        Should NOT set zero address
          ✓ protocol fee recipient (385ms)
          ✓ operational fee recipient (303ms)
          ✓ strategy whitelist (262ms)
        Should NOT deploy
          NFT721
            ✓ if sender is not deployer (259ms)
            ✓ if token name or symbol is empty (352ms)
            ✓ if owner is zero address (114ms)
          NFT1155
            ✓ if sender is not deployer (259ms)
            ✓ if tokens length != amounts length (337ms)
            ✓ if owner is zero address (334ms)
          SocialToken
            ✓ if sender is not deployer (289ms)
            ✓ if token name or symbol is empty (245ms)
            ✓ if owner is zero address (132ms)
          Tags
            ✓ should NOT set tags to ERC721 [sender is not owner of token] (263ms)
            ✓ should NOT set tags to ERC1155 [balance of sender 0] (335ms)
          NFT721
            ✓ should NOT set royalty recipient to zero address (231ms)
            ✓ should NOT set royalty fee (356ms)
          NFT1155
            ✓ should NOT set royalty recipient to zero address (173ms)
            ✓ should NOT set royalty fee (355ms)


    59 passing (28s)
```

We are grateful to have been given the opportunity to work with the Shoyu team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the Shoyu team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.